DOI 10.20357/B7CP4S

BERKELEY LAB

Lawrence Berkeley National Laboratory

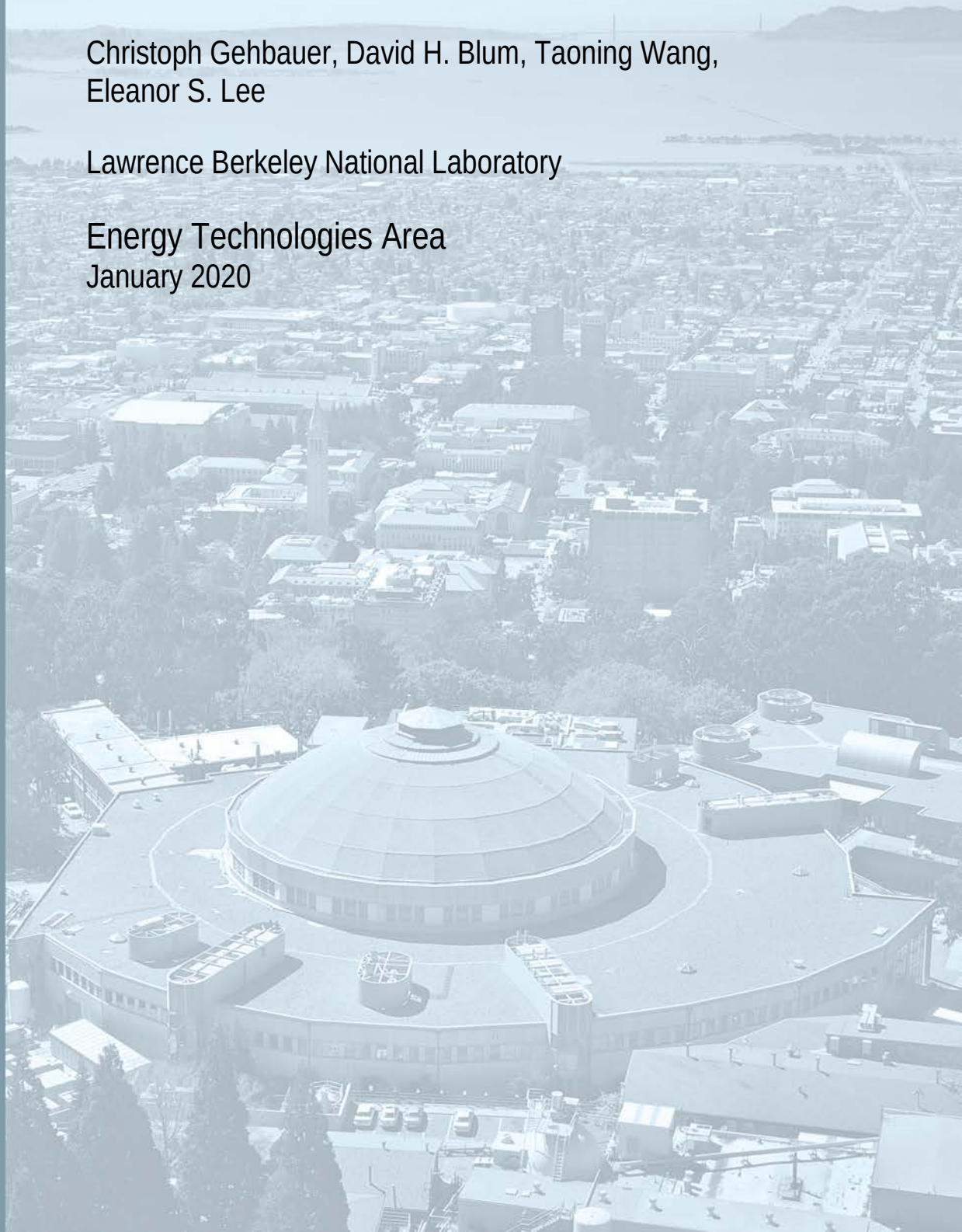
Integrated Dynamic Facade Control with an Agent-based Architecture for Commercial Buildings

Christoph Gehbauer, David H. Blum, Taoning Wang, Eleanor S. Lee

Lawrence Berkeley National Laboratory

Energy Technologies Area

January 2020



Acknowledgments:

This work was supported by the California Energy Commission through its Electric Program Investment Charge (EPIC) Program on behalf of the citizens of California. LBNL is supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Building Technologies Program of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

Integrated Dynamic Facade Control with an Agent-based Architecture for Commercial Buildings

Christoph Gehbauer, David H. Blum, Taoning Wang, Eleanor S. Lee

Building Technology and Urban Systems Division, Energy Technologies Area,
Lawrence Berkeley National Laboratory, Mailstop 90-3147,
1 Cyclotron Road, Berkeley, California 94720 USA

Abstract

Dynamic façades have significant technical potential to minimize heating, cooling, and lighting energy use and peak electric demand in the perimeter zone of commercial buildings, but the performance of these systems is reliant on being able to balance complex trade-offs between solar control, daylight admission, comfort, and view over the life of the installation. As the context for controllable energy-efficiency technologies grows more complex with the increased use of intermittent renewable energy resources on the grid, it has become increasingly important to look ahead towards more advanced approaches to integrated systems control in order to achieve optimum life-cycle performance at a lower cost. This study examines the feasibility of a model predictive control system for low-cost autonomous dynamic façades. A system architecture designed around lightweight, simple agents is proposed. The architecture accommodates whole building and grid level demands through its modular, hierarchical approach. Automatically-generated models for computing window heat gains, daylight illuminance, and discomfort glare are described. The open source Modelica and JModelica software tools were used to determine the optimum state of control given inputs of window heat gains and lighting loads for a 24-hour optimization horizon. Penalty functions for glare and view/ daylight quality were implemented as constraints. The control system was tested on a low-power controller (1.4 GHz single core with 2 GB of RAM) to evaluate feasibility. The target platform is a low-cost ($35/unit) embedded controller with 1.2 GHz dual-core cpu and 1 GB of RAM. Configuration and commissioning of the curtainwall unit was designed to be largely plug and play with minimal inputs required by the manufacturer through a web-based user interface. An example application was used to demonstrate optimal control of a three-zone electrochromic window for a south-facing zone. The overall approach was deemed to be promising. Further engineering is required to enable scalable, turnkey solutions.

Keywords: model predictive controls; switchable windows; dynamic facades; daylighting; windows; building energy efficiency.

Table of Contents

1. Introduction

Dynamic façade technologies such as operable shades and windows, switchable coatings, and daylight-redirecting technologies have significant technical potential to reduce lighting and heating, ventilation, and air-conditioning (HVAC) energy use in the perimeter zones of commercial buildings. The performance of these systems is reliant on active, automated control that successfully balances energy use tradeoffs within comfort or other occupant-related constraints. When successful, integrated dynamic facades can deliver significant energy savings and enable downsizing or elimination of perimeter heating and cooling systems. Use of dynamic facade technologies can also open up the opportunity to use larger area windows, enabling greater access to daylight and views. In a competitive real estate market, where health, wellness, and workplace amenity are becoming increasingly important to occupants, such systems can provide a significant advantage over conventional leased space.

Grid modernization activities in the face of increased use of renewable energy resources, such as photovoltaics (PV), are changing the conventional economic context for building energy-efficiency technologies. With PV now providing a growing percentage of the demand during peak sunny periods in the state of California, electricity rates are in a state of flux and will likely remain variable for some time as industry develops solutions to accommodate intermittent distributed energy resources. As we move closer to the goal of net zero energy buildings, this competitive interplay between renewable energy and energy efficiency solutions, workplace environmental quality, and market economics will continue to change the context for component-level control systems. Integrated, adaptable control systems have the potential to deliver on multi-variable performance claims by responding more optimally to the context of the application even as the context changes, whether at the zone, building, campus, or grid level; however, there are a number of R&D challenges that need to be addressed before such solutions can be broadly adopted.

This study investigates the feasibility of a prefabricated dynamic façade curtainwall unit or retrofit attachment whose control system has been designed to be plug and play and operate autonomously to minimize costs associated with engineering, commissioning, and tuning after occupancy. The approach relies on model predictive controls (MPC), which enable optimization of many variables over a forecasted period, is modular which facilitates scaling from small to large applications, and has the ability to adapt to a changing context over the life of the installation. The key objective of this study as a preliminary step towards a broader vision of integrated building-to-grid control is to assess on a technical basis whether such an approach will ultimately be cost effective, practical, and scalable at the *zonal* level for commercial building applications. The requirements for the conceptual design are laid out (Section 3), then a description of its implementation is given to illustrate the technical issues that need to be solved in order to realize such a solution (Sections 3-6). A prototyped control system for a three-zone electrochromic window was implemented with a low-cost embedded controller to assess whether the design could meet the requisite time step demands of the optimization control algorithm (Section 7). Lessons learned and next steps were derived as a result of limited testing (Section 8). Further research will be conducted to address key lessons learned.

2. Background

Model Predictive Control (MPC) is an alternative method that can meet the challenges of new building control paradigms, diagramed in Figure 1 below. With this method, a model of system operation, along with forecasts of disturbances, is used to predict future performance and optimize setpoint schedules and/or control inputs over a specified time horizon. The solution of the first control step is implemented and the optimization is solved again with updated information (system state and disturbance forecasts) for the next control step.

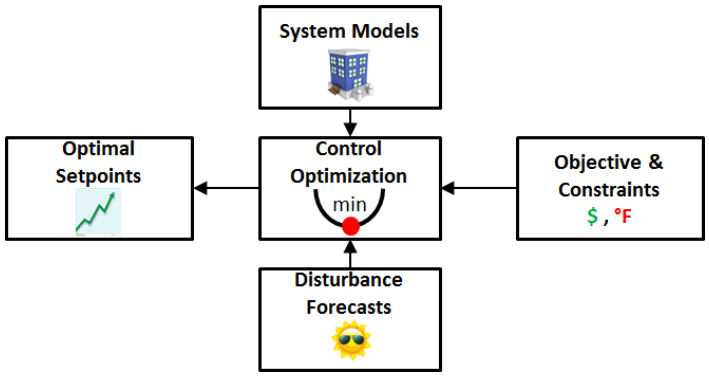


Figure 1. MPC combines system models, disturbance forecasts, constraints, and an objective function into an optimization problem that produces optimal control setpoints.

MPC has historically been developed and used extensively in the process industry [Qin and Badgwell 2003], where refineries and power generators have optimized plant outputs according to predictions of demand and operating conditions. While first considered nearly 30 years ago for buildings [Braun 1990], the last decade has seen a surge in research on MPC applications for commercial buildings, including energy consumption or energy cost minimization [Oldewurtel et. al. 2010, Greensfelder et al. 2011, Sturzenegger et al. 2016, Corbin 2013, Li et al. 2015, Zakula et al. 2015, De Coninck and Helsen 2016a], electric grid integration [Coffey et al. 2010, Pavlak et al. 2014, Vrettos et al. 2014, De Coninck and Helsen 2016b, Blum et al. 2016, 2017], and occupancy schedule prediction integration [Dong and Lam 2014]. For buildings, models of room thermal response, HVAC, lighting, on-site storage, and occupant preferences are combined with forecasts of weather, occupancy, internal load, on-site generation, and grid signals to produce optimal setpoint or other control input trajectories, typically over a 24-hour time horizon. A classic example is in the context of dynamic electricity pricing. Here, forecasts or notifications of high electricity prices can be combined with a model of HVAC system operation to identify the optimal pre-cooling solution of building zones in order to minimize total energy cost while maintaining occupant thermal comfort. In the context of dynamic façade control models of how shading position affects HVAC load and daylight can be incorporated into an MPC that manages total energy consumption of lighting and HVAC while maintaining acceptable glare and room temperature and maximizing daylight.

2.1. Key MPC challenges

Central to MPC are the model and optimization algorithms, which pose a number of challenges. The models must be sufficiently accurate to predict the performance of the system while computationally efficient to be used within optimization algorithms. Furthermore, calculation speed is not the only computational quality of the model of interest. The mathematical structure of the model plays a large role in qualifying what types of optimization solution algorithms can be used and to what degree of efficiency the optimization problem is solved. This includes both speed and convergence to an optimal solution. In general, gradient-based optimization algorithms are more efficient than numerically-based optimization algorithms [Wetter et al. 2016]. However, gradient-based optimization algorithms require continuous, differentiable models. In the context of building operation, where equipment may only operate in discrete states or operating modes may change at discrete times, this continuity requirement is not always achievable.

In addition, the configuration of the model to achieve the desired accuracy is not straightforward. This includes the building of the model structure as well as the tuning of the model parameters. There are three main classes of models that could be used; white-box, grey-box, and black-box. 1) White box models include those that are built from first-principles, often using a large number of, and likely complex systems of, equations and parameters. For a

building, these parameters would be things like wall construction and equipment efficiencies. Detailed building simulation programs such as EnergyPlus and TRNSYS are examples of these models. While they could have high accuracy, their computational complexity limits their use with optimization algorithms due to practical computational requirements. In addition, it is often the case that the large number of detailed parameters are difficult to tune to achieve the desired accuracy. 2) On the other hand, black-box models include those that are configured purely from collected performance data. These models, often making use of neural network structures or other machine learning techniques, are computationally efficient, but require large and diverse sets of data to train to the desired accuracy. 3) Lastly, grey-box models fall somewhere in between. These models are simplified physical models compared to their white-box counterparts, generally having simpler, lower numbers of equations and fewer parameters. A common example of these models for room thermal response is a resistor-capacitor (RC) model. In addition, these parameters can be tuned using real system performance data. This process of model parameter tuning is often referred to as model parameter estimation or training. This could be done using regression, batch optimization [De Coninck et al. 2016], or online filtering [Bonvini et. al. 2014] techniques. Often for equipment, such as lighting and HVAC, performance maps are used with mapping parameters estimated on performance data. A common example is the chillerEIR model from EnergyPlus, which uses a number of polynomial equations to determine the power consumption of a chiller given part-load ratio, condensing temperature, and chilled water temperature. Fan laws are another common example to determine fan power from a given airflow rate.

2.2. Proposed MPC approach for dynamic façades

In a previous MPC-based study [Coffey et al. 2013], models of glare, illuminance, and heat gain associated with a given window and shade configuration were combined with models of room thermal response, lighting equipment, and HVAC equipment to perform offline optimization of window shade position over a large range of operating conditions, including solar conditions and outside air temperature. The result was a look-up table which could be used to control the shade position according to observed operating conditions, but the solution was constrained and somewhat inflexible. The proposed MPC approach in this project looks to improve upon this previous research in a few ways:

- The first is to implement an approach that can produce an optimal solution for shade position, lighting control, and HVAC control that maintains all three occupant constraints in the room simultaneously; temperature, illuminance, and glare. This improves the effectiveness of the solution.
- The second is to implement an approach where the optimal solution can be solved for in real time. That is, at each control time step, a new optimal solution can be produced based on the most up-to-date room state and forecasted operating conditions. This greatly improves the scalability of the solution, as look-up tables require anticipation of operating conditions a-priori, limiting the application of the look-up table to only the cases considered to create the table, including both the model and all of the operating conditions. As grid signals, weather conditions, and system performance change over time, previously generated look-up tables are likely to become inappropriate for long-term use. Additionally, as the number of controlled states and possible operating conditions grows, the practicality of creating a lookup table for all combinations diminishes. Solving the problem in real time more simply allows the system to respond to operating conditions as they arise, and allows the model of the system to be adapted independently of solving the control optimization problem if accuracy declines over time.
- Lastly, the proposed approach utilizes only open-source, free software, which improves access to the approach by other researchers and industry. A previous approach [Gwerder et al. 2013] was able to integrate the above two improvements, though with commercial software MATLAB (for modeling and scripting) and CPLEX (for optimization solver). Using free software reduces the capital cost of implementation.

In order to be solved in real time, the MPC control optimization problem needs to be solved efficiently. In order to aid scalability, the MPC models need to be adaptable to actual performance in different buildings without considerable configuration. While each of these aspects is still a topic of research in the building MPC community, the approach applied here considers each of these requirements.

- The approach utilizes models written in Modelica [Mattson and Elmqvist 1997], an object-oriented, equation-based, open-source computer language developed for the dynamic simulation of engineering systems with multiple physical domains (e.g. thermal, fluid, and electrical).
- The object-oriented aspect of the language allows for encapsulation of models into sub-models (component models) that can be rapidly replaced and edited without changing the structure of the whole-system model.
- The equation-based aspect allows for use of the more efficient gradient-based optimization algorithms.
- The open-source aspect allows for these models to be easily shared among researchers and industry in the form of libraries.
- The control optimization problem using the Modelica models is solved using JModelica (http://jmodelica.org/), an open-source tool for compiling and optimizing models written in Modelica. In addition to control optimization problems, JModelica is able to solve model parameter estimation optimization problems. This, in addition to the use of object-oriented component models, eases the configuration needed to implement the model in different buildings.
- Finally, MPCPy [Blum and Wetter 2017], an open-source python package developed to facilitate the testing and implementation of MPC in buildings, is used to integrate the above-described modeling and optimization approach with data collection, controller actuation, and user interfaces. In addition, MPCPy can be used to setup and solve parameter estimation problems to tune models based on real data using JModelica, or other methods as needed.

While the above approach has many strengths, it also has some weaknesses. For one, the gradient-based non-linear programming (NLP) problem solver currently implemented in JModelica requires continuous and differentiable models. This precludes the use of integer variables, tables, and other complex modeling structures such as if-statements. As will be described in the sections to follow, this limitation has been considered in the implementation of the models and MPC control solutions. In addition, the memory requirement of the JModelica compiler to be installed and run an optimization can be relatively high compared to the processing power of an embedded microprocessor such as a Raspberry Pi. Lastly, the question of hierarchical control among room, building, and campus is not directly answered here. In this study, we focus on a single room MPC approach where it is assumed that setpoints for temperature and illuminance levels are received from a building-level supervisory controller in these cases.

3. Overview of conceptual design

3.1. Design requirements

In prior research, [Coffey et al. 2013] defined a low-cost autonomous façade control system that could be configured by a non-expert using a simple web-based interface yet deliver optimal, integrated control for HVAC and lighting energy use minimization. A similar, more flexible design was envisioned for this study with the objective of minimizing the cost of both constructing the dynamic façade system and configuring its control system as follows:

For new construction, the manufacturer would fabricate the requisite curtainwall unit with operable fenestration components (e.g., electrochromic glazing, and/or motorized shades), power, built-in sensors, and communications interface at the factory. The curtainwall unit's control system would be pre-configured at the factory and assigned to an exact location in the building based on the construction documents and specifications. Once installed at the job

site, the communication and power lines would be plugged in and the control system would start up automatically. In this scenario, the system would be "plug and play", requiring minimal to no user input during installation. Alternatively, the curtainwall unit could be delivered without pre-settings, then configured by the facility manager or end user at the site using a simple web-based user interface. In either case, changes to the configuration would be possible at any time through the web-based interface: applicable models in the control system would be automatically updated in real time. The same concept could be applied to retrofit applications of motorized shades or other dynamic façade attachments, where the power, sensors, and communications could be packaged with the shade to the extent possible at the factory, then configured on site with the web-based interface.

In order to realize this turnkey approach, the system architecture, supporting models, and hardware would need to be designed to be flexible, allowing expansion according to the size of the application (e.g., restaurant or rented office space versus large office building) and enabling simple reconfiguration as operation or usage of the building changes over the lifetime of the installation. Ideally, each controllable façade control zone (e.g., single office with multiple electrochromic windows) would have its own independent controller so that if communications were disrupted, each façade control zone could continue to operate autonomously. A light-weight (i.e., low cpu and memory requirements) software implementation is envisioned to minimize hardware/ embedded controller costs. The design would enable a user with basic technical knowledge to install, configure, and set up the control system quickly to keep initial labor costs low. In addition, end users would be able to make changes with regard to their individual preferences (e.g., thermal, brightness or glare sensitivity).

3.2. System architecture

In order to satisfy the above requirements, the overall control system was designed as an agent-based system. An agent is defined here as an independent, discrete, self-contained software component with a set of characteristics and behaviors that can function independently, but also has the ability to recognize other agents with which it interacts. In the realm of artificial intelligence, agents have the ability to learn and adapt over time and therefore have some form of memory (database). In this case, some but not all agents have an ability to adapt, as described below.

Tasks within the overall control system were split into individual, autonomous operating units and optimized for a specific objective. Each task or agent was implemented as a lightweight application that could run either on individual devices, or combined with other tasks/ agents on a single hardware "platform" (e.g., Raspberry Pi embedded controller platform).

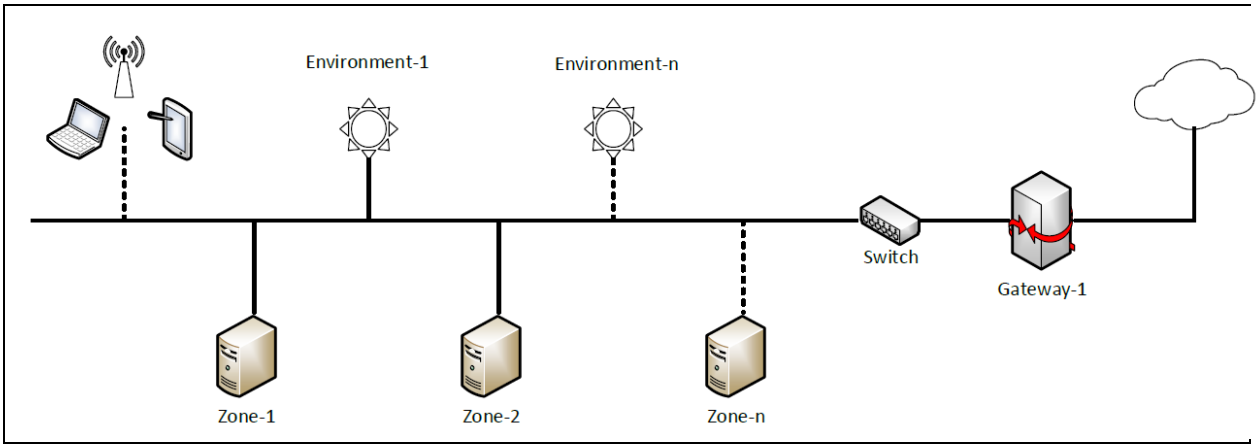


Figure 2. Overall façade control system architecture.

A typical implementation of the overall control system architecture would be as follows (Figure 2):

- The communication, shown as the solid black line, is an Ethernet-based network. Ethernet has the advantage of being a well-documented, reliable and widely available communication protocol that allows high connectivity for end users. Virtually all computers support Ethernet communication: either Ethernet itself or WiFi (IEEE 802.11).
- The dotted black lines indicate extensions of the network that could be an access point for WiFi-connected tablets and smartphones. This provides an option for on-site end user adjustments via the web-based user interface, illustrated on the upper left of the diagram.
- The diagram also shows three zonal controllers as cpus (or embedded controllers) labeled Zone-1, Zone-2, and Zone-n. One zonal controller represents one thermal zone and window orientation (e.g., a south-oriented single office or open plan office area would represent a single zone). In the illustrated example, each zonal controller would run several agents on the same platform. A typical implementation for small scale applications would include agents for zonal configuration and zonal control running on the same platform.
- A gateway, illustrated as a server rack on the right, would be the single point of connection to the internet. It would include agents for zonal mapping and weather forecast. The gateway is also the only device to allow devices to connect to the control network, being the dedicated Dynamic Host Configuration Protocol (DHCP) server for all connected devices. The gateway does not act as a regular gateway by providing an internet connection to underlying devices; it is instead a single node with dual Ethernet connection, e.g., to get weather forecast data from the internet and then share the data in the private control network.
- The last elements for this design are real-time environment sensing stations, illustrated in the upper region and labeled as Environment-1 and Environment-n. In this example, the environment station could include real-time sensing of interior and exterior temperatures as well as solar radiation with an accompanying sensing agent. Generally, all systems are built to be modular and could either be omitted in case of low-end, small scale implementations or replaced if components are already available. One example would be an office building with a central building management system (BMS) which already senses zonal temperatures. In this case, the real-time sensing agent could be adapted to pull data directly from the BMS instead of from a redundant sensor.
- As each device is operating autonomously, the resulting asynchronous communication between devices is realized as simple Hypertext Transfer Protocol (HTTP) GET and PUT standard requests. This also has the advantage that standard web-browsers can communicate with devices which allows synergies for future/ additional applications.
- For this implementation, each of the hardware devices, e.g., zonal controller, environment stations, gateway, is assumed to be a low cost, embedded controller based on a commercially available Raspberry Pi. This microprocessor provides a 1.2 GHz quad-core processor, 1 GB RAM and built-in Ethernet support for a very competitive end user price of $35/unit.

3.3. Control system hardware implementation

Embedded controller/ hardware platform

The system architecture was designed to support lightweight independent agents. Each of the agents were developed as Python modules, which allow simple implementation on virtually any platform; e.g., Linux-based, Windows, MacOS. Generally, most agents were developed for scaling purposes, not computational constraints, and can therefore be implemented on nearly any hardware platform running one of the mentioned operating systems, or run in parallel on the same hardware device. An example of a sample architecture with parallel agent execution was given in Section 3.2. The only requirement for all platforms is that they support Ethernet in order to communicate with other devices. Some agents, such as the dynamic facade control agent, have special computational

requirements for the CPU or memory. In Table 1, a list of all agents and their respective hardware requirements is given.

Table 1. Agent hardware requirements.

Agent:	Hardware requirements:
User interface agent	No additional requirements
Weather forecast agent	No additional requirements
Real-time sensor agent	Hardware requirements to acquire data (dependent on sensor, see Section 4.3)
Real-time solar agent	Hardware interface to connect to a CMOS sensor; platform with processing power on the order of low-end personal computers; e.g., 1.4 GHz Dual Core with 1 GB RAM
Zone mapping agent	No additional requirements
Dynamic façade control agent	Platform with low processing power but increased memory on the order of low-end personal computers; e.g., 1.6 GHz Dual Core with 2 GB RAM

Figure 3. Photograph of the Raspberry Pi 3. Credit: Raspberrypi.org.

The initial implementation of the control system is being tested on a low-cost Raspberry Pi platform (Figure 3). For small-scale applications, the architecture is designed to run *all* of the agents in parallel on this single platform, even though separation of the more computational intense agents would be advised. Technical specifications for the proposed Raspberry Pi 3 are given in Table 2 [Raspberrypi.org, 2017].

Table 2. Specifications for an embedded controller.

Processor	Raspberry Pi 3; Quad Core 1.2 GHz Broadcom BCM2837 64-bit CPU
Memory	1 GB RAM
Communications	Ethernet, BCM43438 wireless LAN, and Bluetooth Low Energy (BLE)
Connectivity	40-pin extended GPIO, 4 USB 2.0 ports, CSI camera port
Storage	Micro-SD slot
Power	5 V/ 2 A max (idle 1.5 W; stress 5 W)
Price	$35 retail per unit

To reduce cost, the control system would need to be designed to decrease computational needs in order to run on a lower-cost platform. One target platform could be a RaspberryPi Zero with a reduced processing power of 1 GHz Single Core and only 512 GB RAM, but with a retail price of only $5 per unit.

In example of the RaspberryPi, the removable micro-SD card would include the actual operating system and Python control agents. Eventual OS or any control system updates could be pushed to the device without any action required by the end user/ facility manager. It also does not need any regular maintenance of software or hardware components, once set up. However, in case of technical failure, the micro-SD card could simply be switched out to a newer version or if the hardware fails, the controller could simply be replaced as a plug-in device, by the facilities manager or a user with basic technical knowledge.

Power distribution

The current control system is anticipated to be powered by theoretically any source using internal DC converters to power the RaspberryPi control system with 5 V_{DC}. However, a low voltage DC system would be desirable to

minimize electric wiring effort, reduce safety hazards, and allow the integration of renewable distributed energy resources (DER) such as photovoltaics (PV) in combination with electric storage. Optimally, the supply voltage of the curtainwall unit would be the same as the one necessary to power the dynamic facade device itself, commonly 24 V_{DC}. A DC-DC converter could step down the voltage to the 5 V_{DC} for the control system.

The concept of a self-powered curtainwall unit could significantly reduce cost and this concept has been investigated in prior research, given that windows are exposed generally to high incident solar radiation. The RaspberryPi would be suitable for this envisioned concept as it has very low power requirements with a maximum 5 W during high CPU load and less than 2 W when idle. Switching of the shading device itself is less critical as operation usually occurs during times when incident solar is high, coincident with peak PV production. Batteries could be used to store excess PV production during the day to keep the controller powered during the night and provide additional power for switching operations of the dynamic façade.

Sensors

The current design was developed to minimize sensor requirements. At present, the recommended basic setup includes real-time temperature sensors to measure indoor and outdoor dry bulb temperature and an occupancy sensor to enable temperature or glare setbacks while zones are unoccupied. For the real-time solar data, the sensor requirements are currently under development. What is envisioned is a low-cost high dynamic range imaging system with sufficient resolution and range to enable adequate characterization of the hemispherical luminance distribution seen by the window. This system is under development by multiple groups around the world and is likely to be available in the near term.

4. Supporting agents

The actual software implementation is based on open-source software tools and can run on any operating system. Most agents are built as a lightweight implementation in Python in combination with a simple Python-based web server and database. In this section, the following supporting agents are described in detail (Table 3). These agents support the operations of the dynamic façade control agent, which performs the necessary calculations and optimization, then executes real-time control of the dynamic façade element(s) (described in Sections 5 and 6).

Table 3. Agents and associated task objectives

Agent:	Task objective:
User interface agent	Web-based interface that accepts site-specific inputs from end users
Weather forecast agent	Obtains real-time and forecast data from the local weather station
Real-time sensor agent	Obtains real-time data from environmental sensors
Real-time solar agent	Obtains real-time data from a window-mounted sensor
Zone mapping agent	Maps data from the web-based interface to specified zones in the building application

4.1. User interface agent

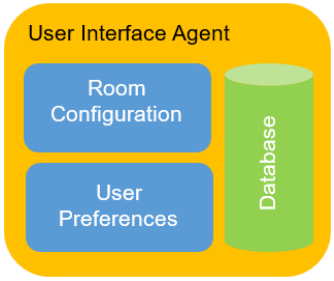


Figure 4. Diagram of the user interface agent.

A web-based user interface was designed to accept simple inputs in order to a) initially configure the overall control system during commissioning, and b) allow the end user to input their personal preferences for control such as brightness and glare sensitivity. The agent is illustrated in Figure 4.

The entire user interface is implemented in Python to enable greatest flexibility in actual implementation. The end user can access the interface via any web browser on a computer, tablet or phone and connect to the dedicated interface by entering either the zone's name or IP address (e.g., http://zone1). The agent's Python-Django module responds to a user request by rendering a web page containing the various inputs necessary to set up the control system. Depending on the facility manager or owner's requirements, occupant(s) could be permitted access to the web-based user interface to adjust a limited subset of settings to preferred levels. Both the room configuration and user preferences are stored in a simple Python-based database and are available via a backend HTTP communication to other agents. All changes to the control configuration are logged and can be accessed to optimize or diagnose the controls, in the case of ongoing occupant complaints. An initial web-based implementation that provides interactive real-time visualizations and feedback to the end user via an iPhone-sized device is shown in Appendix A. A summary of the inputs needed by the current design is given in Table 4.

Like Coffey's original prototype, the inputs for the initial set up are designed to be minimal to reduce commissioning costs, both in terms of technical expertise and time, and is envisioned to be configured by the manufacturer or the facility manager (Figure 5). This raises the logical question of how accurate can the control be if the details of the building are not accurately represented in the model. As indicated in Section 2, the approach taken in this study is to use grey box models to predict HVAC and lighting loads and occupant comfort for conventional cases (e.g., sidelit perimeter office), which are then adapted or calibrated based on feedback from environmental sensors or the occupants.

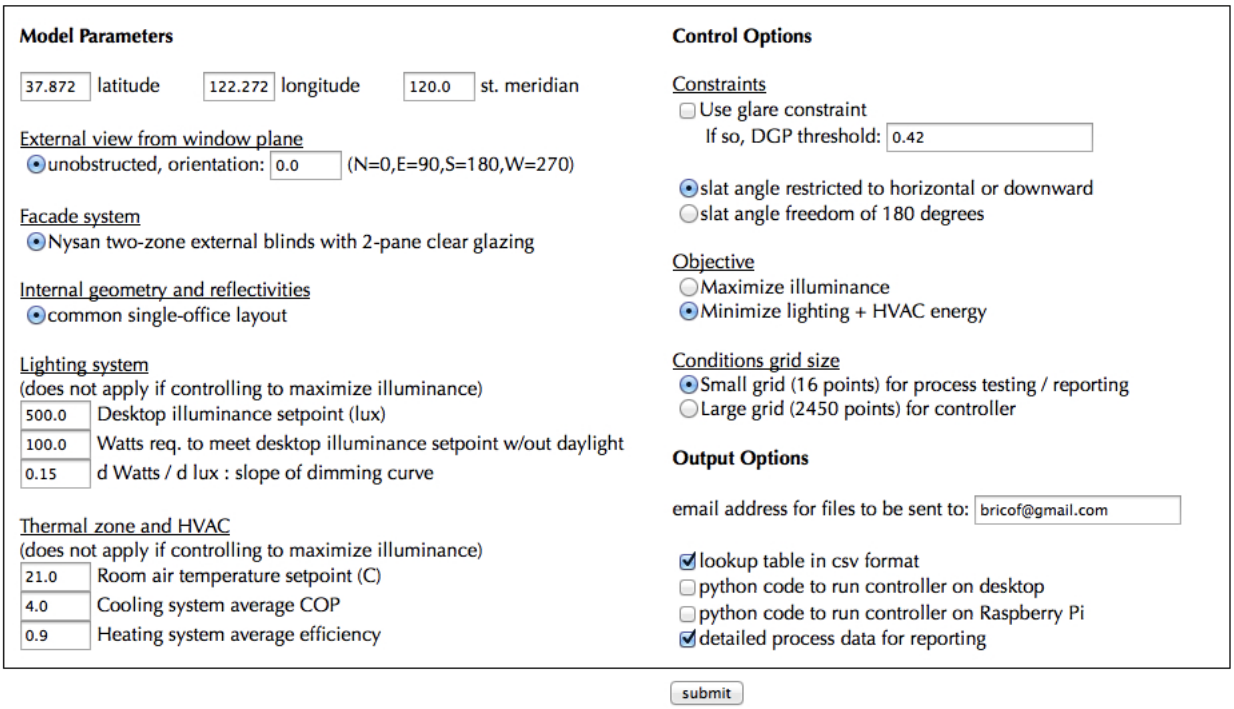


Figure 5. Example web-based interface.

Table 4. Inputs required by the new web-based interface (Appendix A).

Information input by the manufacturer or facility manager

- Site information (zone ID, state, city, orientation)
- Building (construction year, construction type)
- Room dimension (width, height, depth)
- Window dimension (width, height, sill height and reflectance, depth, setback, number of windows, orientation)
- Dynamic facade system (type, number of horizontal control zones)
- Lighting system (zone location relative to the window, lighting system type, setpoint)
- HVAC system (cooling type, cooling efficiency, heating type, heating efficiency)
- Occupant positions (view direction, orientation)
- Occupant preference (brightness, glare)

Information input by the occupant of the space

- Room brightness (as a slider-bar for 5 different levels)
- Glare sensitivity (as a slider-bar for 5 different levels)
- One-time override? (checkbox for yes/no)
- Comment for change (text input)

4.2. Weather forecast agent

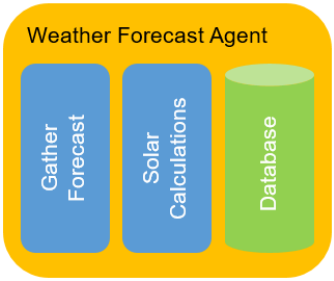


Figure 6. Diagram of the weather forecast agent.

Weather forecast data are used to determine the projected HVAC and lighting loads, which is necessary to determine the optimal operational solution for time-dependent technologies and strategies; e.g., load shifting with thermal mass, switching delays of dynamic façade components, variable energy tariffs. Inputs can include any data available from the local weather station; e.g., solar irradiance, outdoor temperature, wind speed and direction, relative humidity, etc.

A weather forecast agent (Figure 6) was designed to obtain the hourly, seven-day weather forecast from the National Oceanic and Atmospheric Administration (NOAA). The agent was designed to be modular so third-party weather forecasts could be added if available. To get the forecast, the agent connects to the user interface agent to obtain site information (e.g., latitude and longitude). NOAA provides an application programming interface (API) where forecasts can be queried via HTTP GET requests by providing the site's location. The weather forecast agent is currently designed to retrieve only the outdoor dry bulb temperature and sky cover forecast, but additional points of data could be added if needed.

The received temperature forecast can be used as provided and is therefore stored in the agent's database directly.

The sky cover forecast is not useful in its current form and is therefore converted to solar radiation within the agent. The current implementation accomplishes this step by utilizing publicly available, site specific Typical Meteorological Year (TMY) files [NREL 2017] to generate solar baselines for each month. The baselines are then multiplied by the sky coverage to obtain actual horizontal solar radiation levels. Equations 1 and 2 show the conversion process from percentage sky coverage, s, to global horizontal irradiance (GHI) and diffuse horizontal irradiance (DHI), both in units of W/m^2:

$$GHI_{actual} = GHI_{baseline} (1 - 0.75\, s^3) \quad (1)$$

$$DHI_{actual} = DHI_{baseline} (1 - s^5) + GHI\, s^5 \quad (2)$$

To convert the GHI and DHI to direct normal irradiance (DNI), the time of day and site specific sun altitude, θ, is taken into account. Equation 3 shows this conversion (also in W/m^2):

$$DNI = (GHI - DHI) / \cos \theta \quad (3)$$

The data from NOAA is currently pulled every 5 minutes, but observations showed that changes in forecast usually occur on an hourly basis so activity on the network could be reduced. Once all solar data are converted, it is stored in the internal database to enable use by other agents.

4.3. Real-time sensor agent

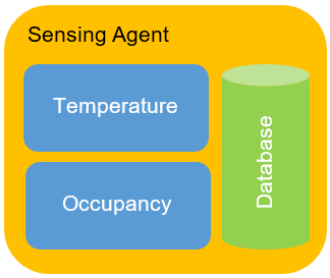


Figure 7. Diagram of the real-time sensor agent.

Real-time environmental data gathered from sensors can be used to improve model predictions and attain more optimal control. Sensors could be located within the framing construction of the curtainwall unit or window attachment, within the room, or made available from other sensor networks through access to the building management system (BMS). The sensor agent acquires sensor data through the local network then performs the necessary conversions and data processing to derive the final data (Figure 7). Data conversion and sensor calibration coefficients can be stored within the agent and updated through the web-based interface. The frequency of data acquisition can be set as a configuration parameter in the database and is theoretically only limited by the conversion time of the embedded controller or latency of communication protocol. A standard setting would be a one-minute interval. All sensor data are stored in a local Python database and are available to other agents via HTTP requests.

Most embedded microcontrollers provide multiple analog and/or digital inputs natively. In the case of the Raspberry Pi, there are 26 general-purpose input/output (GPIO) ports available. For digital sensors, such as an occupancy sensor, inputs can be acquired directly from the sensor. For analog sensors such as a thermistor for reading temperature, an analog-to-digital converter (ADC) board may be needed. Some sensors come with a built-in ADC, which enables a direct connection to a Raspberry Pi. The GPIO ports could be used for other sensing devices such as light or infrared sensors or could be used to enable communication to an already existing monitoring device (e.g., BMS). For a BMS, the GPIO pins could be configured to emulate for example a serial communication protocol to pull information directly from the BMS without involving additional sensing.

4.4. Real-time solar data agent

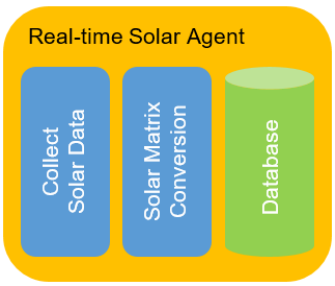


Figure 8. Diagram of the real-time solar data agent (left) and example image of a window-mounted sensor (Credit: https://www.indiegogo.com/projects/geckoeye-security-for-peace-of-mind#/)

For the models used in this study, a window-mounted sensor was needed in addition to the weather forecast data in order to obtain more accurate, reliable control. A real-time solar data agent was developed to acquire hemispherical luminance data at the plane of the window, convert this data to a text file, then store this file in the internal database to enable use by other agents (Figure 8). The total time to collect the data and convert the data into the text file takes approximately 4 minutes so a sampling rate of once every 5 minutes is achievable with the current configuration.

4.5. Zone mapping agent

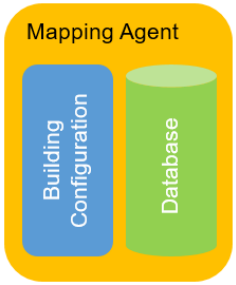


Figure 9. Diagram of the zone mapping agent.

The zone mapping agent (Figure 9) assigns the IP addresses of the supporting agents to the specific thermal zone in the building. This enables the dynamic façade devices to power up, ask the zone mapping agent for the configuration of dependencies, then start operating with a self- commissioning, plug and play approach. The zone mapping agent is the single, central configuration agent for all other supporting agents and zone controller agents. The current zone limit is 255, which is the maximum number of nodes for IPv4. This limit can be expanded by adding a standard Ethernet router and making minor adaptions in the mapping agent to include the additional IP address slots.

The zone mapping agent has a simple user interface to assign necessary inputs (IP addresses of the supporting agents) to specific zones. The mapping agent has a fixed IP address, e.g. 122.168.1.2. The IP addresses of the supporting agents can be determine a priori from the embedded controllers used to implement the agents. This configuration, while time-consuming, cannot be automated unfortunately.

As an example of mapping, the weather forecast agent relies on site information entered in the user interface agent. Here, the zone mapping agent would provide the address of the specific user interface agent to the weather forecast agent. Typically, the zone mapping agent would be located on the gateway hardware.

5. Dynamic facade control agent

The objective of the façade controller is to minimize energy use or energy cost while maintaining occupant comfort within acceptable bounds. The façade controller does this by computing at each time step the window heat gains, daylight illuminance within the zone, and level of visual comfort over a 24-hour optimization horizon. The zone's projected heating, cooling, and lighting energy use are computed from these data, then the optimal state of the dynamic façade is determined using an advanced optimization solver with visual comfort as a constraint. The current control system design enables autonomous façade control without the need for data from or control of other related building systems (i.e., lighting and HVAC). The solution could be scaled to enable active control of other systems, such as adjustment of the thermostat or dimmable lighting system, but for the purpose of this discussion, the focus of this initial solution is on façade-only zonal control.

The proposed model-based approach is a hybrid solution where real-time data are used with forecast data to arrive at the dynamic façade control state on a time step basis. The forecast data (i.e., window heat gains, daylight, comfort) are generated using the 24-hour forecast weather data as input to the model predictive control (MPC) algorithms. An additional, more accurate calculation is made using real-time sensor data with the same MPC algorithms. The two datasets are used with the optimization solver to derive the final façade control state. The motivation behind this hybrid approach is principally to address occupant comfort. Discomfort glare is highly dependent on real-time, site-specific solar conditions. Prior research revealed that comfort levels based on weather forecasts gathered at nearby airports or weather stations were insufficiently accurate to satisfy occupant requirements. These real-time data are used in the optimization solver to improve accuracy of control. The data can also be used to improve accuracy of the models.

In general, the challenge of solving an optimization problem is that the MPC calculations and optimization have to be achieved within the desired control time step and limited resources (i.e., cost constraints) of the embedded controller (e.g., Raspberry Pi). If the control time step is too long (e.g., 15 minutes), the end user is unlikely to accept discomfort for this length of time. To minimize discomfort, the control time step must be significantly reduced. Tests of the control system implementation were conducted on the embedded controller to evaluate feasibility of the implementation, then the models were adjusted to meet the time and resource constraints. These tests will continue as the models are refined and tested against accuracy requirements. Design assumptions regarding simplicity of the model, methods to adapt the model based in environmental data, and hardware constraints will be modified over an iterative design-evaluation process. Currently, a 5-minute time step is feasible on a platform such as the Raspberry Pi. Further improvements to the algorithms could reduce the time step down to 2-3 minutes.

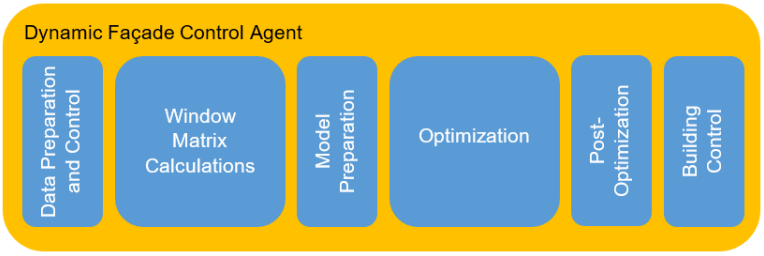


Figure 10. Diagram of the dynamic façade control agent.

All calculations needed to determine the optimal state of the dynamic façade in each zone are defined within the dynamic façade control agent, which communicates with all of the supporting agents mapped to the zone. The agent uses open source Radiance, Modelica, and JModelica tools, all wrapped in a Python framework, in a multi-step process to compute the optimal state of the dynamic façade control elements (Figure 10). In the following sections, each of the steps of the calculation are described in detail and in chronological order of the control operation. To evaluate initial feasibility, an example dynamic façade was defined for testing the control logic both in simulation and for initial field testing. The façade consisted of an electrochromic (EC) window subdivided into three horizontal sections, each independently controlled to one of four tint states (64 states in total). This example application is described within the context of the optimization process. Note however that the underlying models are applicable to a wide variety of specular and optically-complex fenestration systems.

5.1. Data preparation and control

The dynamic façade control agent is called every control time step (currently every 5 minutes) where the process of optimized control starts. In this first step, data from the various supporting agents are queried and checked for consistency, which involves verifying that the agents are providing recent data within a specified timeframe. All

inputs for the dynamic façade control agent (including the optimization) are derived from the following supporting agents:

- User interface agent (all inputs listed in Section 4.1)
- Weather forecast agent (currently outdoor temperature and solar forecast for 24-hour optimization horizon)
- Real-time sensor agent (currently real-time indoor and outdoor temperature, occupancy)
- Real-time solar data agent (hemispherical luminance data)

Once all data have been received and checked for consistency, the agent then checks to see if basic user inputs (e.g., occupant position) have been changed. If there have been no changes, then the normal operation continues to process the input data and transfer the data to the next step of the calculations: window matrix calculations. Even though data are spread out over several devices, HTTP latencies are fairly low and the amount of transmitted data is very small. The duration of this whole process takes only a few seconds. If there was a change in the user input, then all user inputs are reformatted and a new control set up is initiated via the window matrix calculations.

5.2. Window matrix calculation

A three-phase matrix calculation method was used to compute window heat gains, daylight illuminance, and discomfort glare within the zone [Klems 1994, Ward et al. 2011]. This method relies on a backwards ray-tracing software tool called Radiance [Ward and Shakespeare 1998] to derive two matrices that quantify angle-dependent incident (from the sun and sky) and outgoing (into the room) solar radiation at the plane of the window for the specific geometric configuration of the zone and window (defined through the web-based user interface). The third matrix is defined by the angle-dependent properties of the façade system itself, called bidirectional scattering distribution functions (BSDF), which define the intensity of outgoing solar irradiance over a specified solid angle for a paired incoming and outgoing direction of solar irradiance.

For window heat gains, the matrix calculation produces the total sum of transmitted, reflected, and absorbed solar irradiance from the sun, sky, and ground hemisphere seen by the window, which is then used in the window heat balance calculation to determine total window heat gains. Alternatively, the total solar irradiance is multiplied by an angle-dependent solar heat gain coefficient (SHGC) to determine total window heat gains. Conductive window heat gains are currently not included in the calculation since it is not a parameter that is affected significantly by control of the dynamic façade elements in commercial building applications.

The matrix calculation also produces daylight illuminance, which is calculated for a user-specified grid of points at a specified workplane height above the floor. Lighting energy use is derived later (see Section 6.2) from an average of these data for each of the control zones specified in the user interface. The lights are assumed to be continuously dimmable, with light output to power dimming curves that are either defined by the generic configuration of the installed hardware (selected from a pull-down menu from the user interface agent) or specified by the lighting consultant.

Discomfort glare can be computed based on rendered images of the luminance distribution from the view point of the occupant or more simply from vertical illuminance at the eye. Initial tests to produce rendered images on the embedded controller took over three days to compute. This was deemed impractical, especially if the occupants are permitted to change their task view point using the web-based user interface. The latter method was selected to reduce computation time. The matrix calculation produces vertical illuminance, which is used to compute glare level in subsequent calculations (see Section 6.4). A hybrid method using both image and illuminance data is being considered to improve control accuracy.

The current implementation was designed to enable modeling of a rectangular box-shaped room with a single sidelit window with dimensions and properties specified through the user interface agent. This space type represents the majority of spaces in most commercial buildings. The room geometry is used by the control agent to generate the first two matrices within Radiance for each subdivision of the window (three zones for the electrochromic window use case). As indicated in the prior paragraph, generation of these matrices can take time when computed using the limited cpu and memory capabilities of a low-cost embedded controller. With the EC use case, it took about 25 minutes to compute the matrices for three EC zones, 25 interior workplane illuminance sensor locations, and three view locations within the room. The matrices are generated once and are recomputed only when the zone is reconfigured through the web-based user interface.

The BSDF matrix for each state of the device (four tint states of the EC window) is defined by the manufacturer or architectural/ engineering consultant (if used for energy code compliance, for example) using either the Radiance *genBSDF* tool and/or the WINDOW software [Mitchell et al. 2008]. These matrices are loaded into the agent's database prior to execution of the calculation. Once the control agent is initiated, the appropriate BSDF matrices are retrieved for use in the three-phase calculation.

The three-phase calculation is then executed for each of the states of the dynamic façade (four tint states for the EC window) and each of the forecast time steps (24 one-hour time steps) using the solar radiation data retrieved from the weather forecast agent. A similar calculation is performed based on real-time solar data retrieved from the real-time solar data agent, but only for the current time step. Results for these discrete control states are output in a tab-separated value file to a database. Currently, this forecast and real-time calculation takes about one minute per time step.

5.3. Model preparation

After the computation of window-related performance data, further preparations are required prior to optimization: a) retrieval of data on the current (real-time) state of the dynamic façade and zone temperature, and b) conversion of window heat gain, daylight illuminance, and discomfort glare data from discrete control states to an analytic data set for use by the optimization solver.

For step a), the current dynamic façade state and room temperature are retrieved from the supporting agents. The current façade state can be read via the real-time sensor agent, if measured, or retrieved from the previous time step's control command. The current room temperature is available from the real-time sensor agent and is passed to the optimization.

For step b), the conversion of window-related performance data for each discrete façade state or position to continuous analytic functions which represent the states as continuously variable is necessary because the Modelica-based optimization solver, JModelica, uses a non-linear programming algorithm from IPOPT [https://projects.coin-or.org/Ipopt] that cannot accept mixed-integer variables. In general, optimization problems can be solved most efficiently if derivative information can be used to ensure the conditions of optimality are met. However, this information becomes more difficult to evaluate when there are discontinuities in state variable functions and integer restrictions on control variables.

Depending on the type of dynamic façade, the fitted analytic function can vary. Electrochromic windows are optically very simple compared to light-scattering systems such as motorized venetian blinds, for example. For the EC use case, the tint level as a function of tint step, u, is exponential $f(u) = A^{(B*u)}$, with A and B being constants. An example of this analytic function can be seen in Figure 11. For this electrochromic window, tint levels are defined by center-of-glass visible transmittance (Tvis): state 1: Tvis=0.01, state 2: 0.06, state 3: 0.18, and state 4: 0.60.

These values were converted to a relative Tvis scale from 0-100%, with 0% corresponding to opaque and 100% corresponding to the clearest available state (state 4).

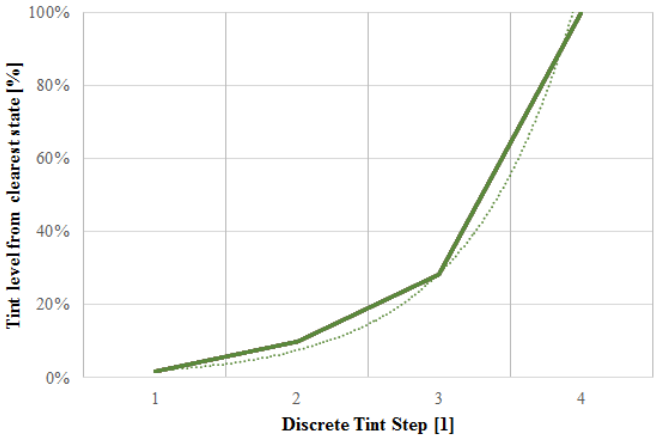


Figure 11. Exponential fit (dotted line) of tint step to relative visible transmittance (0-100% of tint range) for an electrochromic window with four discrete tint states (solid line).

The constants A and B are regressed from the results of the window calculation for each variable, y, and for each of the 24 one-hour time steps, where y is window heat gain, daylight illuminance, and glare. The regressed value of each variable, y, at a given time step (t) can be calculated as a function of the tint step, u, and clear glass value, y_{max}, as shown in equation 4 below:

$$y(t) = f\{u(t)\} * y_{clear}(t) \quad (4)$$

Once the regressions have been completed, the input data is passed to the optimization solver – the total process including the regressions takes several seconds. The data can be divided into four basic sets of inputs, where each input is given as a time series over the forecast 24-hour period (e.g., 24 data points with hourly time steps):

- Forecasted inputs: incident solar radiation, outdoor dry-bulb temperature, window heat gain for each control zone, daylight work plane illuminance for each control zone (WPI), and vertical illuminance at the occupant's view position(s)
- Constraints: WPI minimal light level, maximum glare value, indoor air temperature, and operational constraints on the dynamic façade device (e.g., EC switching time, roller shade positions, etc.)
- Initial values at start of optimization: current shade position, current temperature

6. MPC models and optimization

The dynamic façade control agent was designed to use existing open source tools to conduct the final step of controls optimization. The model predictive control (MPC) models are implemented in Modelica, then the controls optimization problem is solved using JModelica for each time step. The Modelica model used for each zone is shown in Figure 12.

Six sub-models are contained within this model:

- Zonal thermal response
- Lighting power consumption
- Window performance

- Discomfort glare
- HVAC system/ power consumption
- Non-energy penalties for discomfort glare and lack of view and daylight

Inputs to the model include:

- Outdoor dry bulb temperature (current and 24-hour forecast)
- Cooling control signal
- Heating control signal
- Lighting control signal
- Inputs related to the dynamic façade (inputs for each window control zone):
    - Current dynamic façade state
    - Window heat gains for maximum transmittance state for the current and 24-hour forecast period (clear EC glass)
    - Daylight workplane illuminance for the maximum transmittance state for the current and 24-hour forecast period (clear EC glass WPI)
    - Vertical illuminance at the eye for maximum transmittance state for the current and 24-hour forecast period (clear EC glass)

Outputs from JModelica include:

- Control signal for each of the dynamic façade control zones (e.g., tint state)

Outputs from the Modelica model include predicted values as a result of control:

- Zone dry bulb temperature
- Workplane illuminance
- Discomfort glare level
- Lighting power level
- Cooling power level
- Heating power level
- Total electric power level (sum of lighting, heating, and cooling)

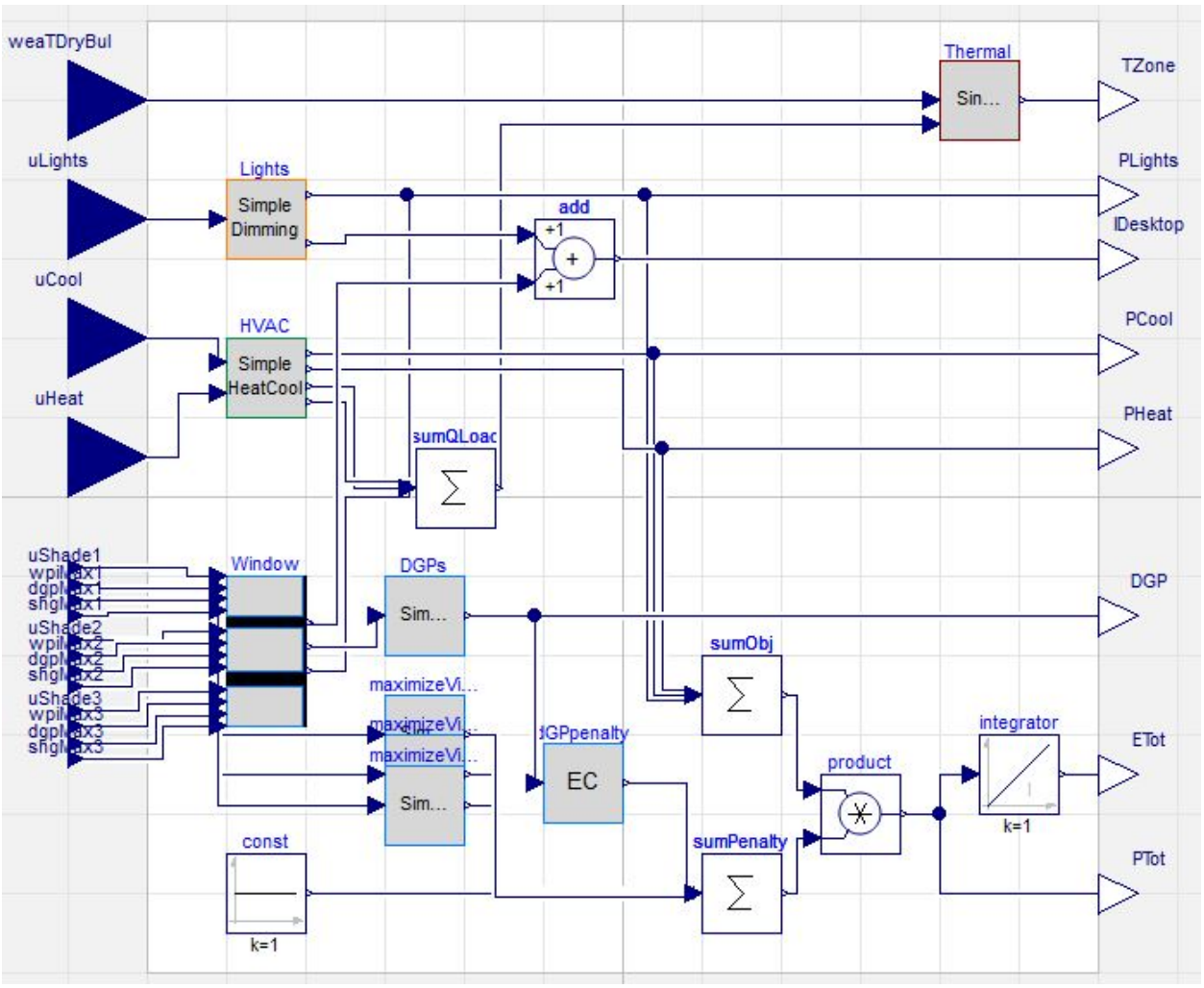


Figure 12. Block diagram of zone model written in Modelica.

6.1. Zone thermal response sub-model

The Modelica zonal thermal response model is a single resistance, R (e.g., 0.05ºK/W), and capacitance, C (e.g., 100 kJ/ºK), with the state equation for such a model presented by equation 5. The total zonal heat gains and outdoor dry bulb temperature, $T_{outdoor}$, are inputs and the zonal temperature, T_{zone}, is calculated. The zonal heat gains are the sum of lighting power, $P_{lighting}$, window heat gains, q_{window}, HVAC heating, $q_{heating}$, and HVAC cooling, $q_{cooling}$. This model approximates the thermal mass of the zone while being very simple for control optimization purposes. More complex RC models could be used in the future for more accurate zonal thermal response.

$$C\frac{dT_{zone}}{dt} = \frac{(T_{outside} - T_{zone})}{R} + P_{lighting} + q_{window} + q_{heating} - q_{cooling} \quad (5)$$

6.2. Lighting power sub-model

The lighting power model, presented in equation 6, is a simple linear implementation of the efficiency, $\eta_{lighting}$, of lighting output on WPI, e.g., 0.3 W/lx$_{WPI}$. The input signal, $u_{lighting}$, which is between 0 and 1, is multiplied by a

WPI capacity of the lighting system, WPI_{max}, to determine the actual WPI output. A minimal lighting level based on hardware constraints or emergency lighting can be implemented as a limited low-end power output, e.g. 10 %.

$$P_{lighting} = \eta_{lighting}\, u_{lighting}\, WPI_{max} \quad (6)$$

6.3. Window sub-model

The window model implements the results of the window matrix calculation assuming shade states are continuous through the operating range. This is for compatibility with the control optimization solver as explained in the previous section. For each control zone of the window, the actual illuminance, total heat gain, and glare are calculated as a function of dynamic façade state or position control signal and the maximum "clear-glass" values of these variables through time, which are inputs to the model as described previously.

For the use case on electrochromics, there can be a substantial delay between setting the tint state and the EC window achieving the tint state. To accommodate this delay, a first order filter with a fixed time-constant, e.g., 15 minutes, has been implemented between the control signal input and the actual façade tint state used to calculate actual window heat gain, WPI, and glare. The actual WPI from daylight is added to the contribution from the lighting system to determine the total WPI. The actual window heat gain is included as part of the zonal heat gains in the zonal thermal response model.

6.4. Glare sub-model

The discomfort glare model is an implementation of the simplified daylight glare probability index (DGPs) where glare is correlated to vertical illuminance at the eye (Iv) [Wienold 2009]. This simplification neglects the influence of individual glare sources and therefore can only be applied if direct sun is not in the field of view (equation 7).

$$\text{DGPs} = 6.22\ 10^{-5} * \text{Iv} + 0.184 \quad (7)$$

6.5. HVAC system model

The Modelica HVAC system model, presented in equations 8 and 9, implements constant efficiency for heating, $\eta_{heating}$ (e.g., 0.9), and coefficient of performance for cooling, COP (e.g., 3.5), to determine heating and cooling electrical power from heating and cooling control signal inputs, $u_{heating}$ and , which take values between 0 and 1. This control signal is scaled by the capacity of the heating and cooling systems, $q_{heating,max}$ and $q_{cooling,max}$ to determine the heating and cooling electricity use to be included in the zonal thermal response model.

$$P_{heating} = \frac{u_{heating} q_{heating,max}}{\eta_{heating}} \quad (8)$$

$$P_{cooling} = \frac{u_{cooling} q_{cooling,max}}{COP} \quad (9)$$

6.6. Non-energy penalties (soft constraints)

Attaining energy savings is usually secondary to maintaining occupant comfort and to some degree a base level of indoor environmental quality. Therefore, additional soft constraints were implemented, which add penalties to the model's cost function (e.g., total energy consumption) to force the optimization in certain directions (e.g., to minimize glare). These constraints usually add complexity as the physics-based, balanced energy consumption model is modified by arbitrary offsets, so the magnitude of the constraints have to be chosen with careful

consideration of the impact on the optimization result. In this specific framework, two soft constraints have been implemented to date: a) penalty for discomfort glare, and b) penalty for decreased daylight/ view.

Penalty for discomfort glare

The first is an exponential multiplier of apparent glare to incentivize the model to stay at low glare levels. This modification is necessary as optimization results may choose the clear tinted state (in the use of the EC window) to maximize solar heat gains, especially during heating periods, with glare being maintained just below the maximum limit (e.g., DGPs of 0.4). This would result in maximum energy savings but would push glare levels to "perceptible" levels. Lowering the DGPs limit (e.g., DGPs of 0.35 or "imperceptible" levels) is also not desirable as physical limitations of the dynamic façade may provide insufficient control during peak periods even at its most blocking or darkest state, which would lead to infeasibilities in the optimization. Implementation of glare as a soft constraint is shown in equation 10 and Figure 13, where the output is chosen to be between 0 to 1 for a DGPs range from 0 to 0.45 ("disturbing" visual discomfort) as hard constraints.

$$\text{Penalty}_{DGP} = 5 * 10^{-5} * \exp(22 * \text{DGPs}) \quad (10)$$

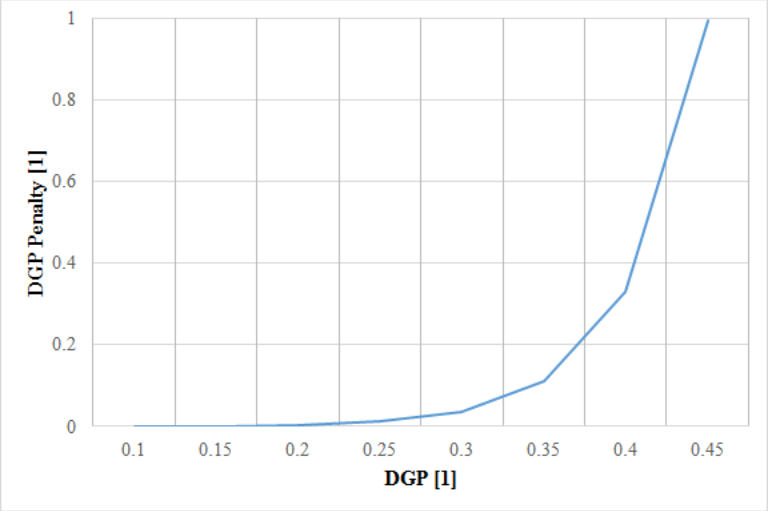


Figure 13. Penalty function for discomfort glare.

Penalty for reduced daylight and view

The second implementation is a view penalty which implements an occupant preference for natural daylight and outdoor views versus electric lighting and dark tinted EC windows. The implementation is a simple linear penalty for each of the window zones, where the darker relative tint level of the EC window zone (Tvis' between 0-100%), the higher the penalty (equation 11):

$$\text{Penalty}_{View} = \text{sum over all window zones } [(\text{Tvis'}_{Clear} - \text{Tvis'}_{Actual}) * \text{Penalty}] \quad (11)$$

The penalty is chosen so that it scales from 0 to 1 for the available tint states.

The penalties are summed up and added to the model's cost function, which is total energy consumption, by scaling the time step's energy consumption from 0-100% penalty for each of the soft constraints. The sensitivity and outcomes of these assigned penalties will be investigated in future work.

6.7. Optimization

The MPC model and inputs are supplied to JModelica to perform the control optimization through the use of MPCPy. MPCPy is an open-source, python-based framework that facilitates MPC implementation [Blum and Wetter 2017]. This includes collection of input data, training of MPC models, and solving of control optimization problems.

Optimization variables are the lighting control signal, shading control signal for each of the window control zones, the heating control signal, and the cooling control signal. Only the dynamic façade shading control signal is used as an outcome of this optimization at this time. It is assumed that local loop controllers on the HVAC and lighting systems will maintain temperature setpoints and illuminance levels respectively. The exogenous input variables are the exterior dry-bulb temperature and clear-glass illuminance, window heat gain, and glare values for each of the three window sections (for the 24-hour optimization horizon). Hard constraints are applied to the zonal temperature and WPI setpoint level in the form of time series. These constraints can be time-varying and modified in future phases of the project according to expected occupancy schedules and preferences. Glare and view penalties are implemented as soft constraints as described in the previous section. The objective of the optimization is to minimize the total energy consumption (HVAC and lighting) while maintaining acceptable glare and view levels over a 24-hour horizon with a forecast interval of one hour.

The process for optimization in MPCPy and JModelica is summarized in Figure 14 below. MPCPy takes the model modelica file (.mo file), the objective definition, and the constraint definitions and produces an Optimica [Akesson et al. 2010] file (.mop), which is an extension of Modelica used to define optimization problems. The .mop file, along with any optimization options, is delivered to JModelica, which translates the Optimica model code into C-code, where it is then prepared by CasADi [Andersson 2013] for solving in IPOPT. Specifically, CasADi performs direct collocation [Magnusson and Akesson 2012] on the model in order to prepare a non-linear programming problem (NLP) to be solved by IPOPT's non-linear programming solver.

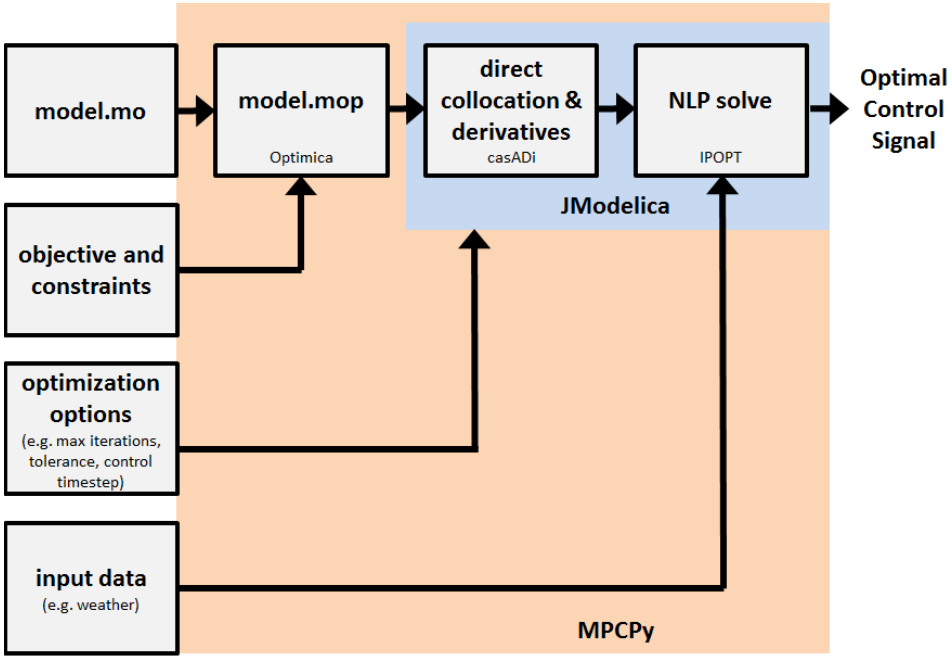


Figure 14. Optimization process using JModelica.

The solver is configured to a maximal time window for the optimization of 120 seconds, to ensure that the dynamic facade control can meet its real-time deadline of 5 minutes per time step. If the optimization exceeds this time limit, then it stops with an error message that the optimization time limit was reached. This could happen in rare cases when the actual solar radiation conditions are unfavorable and the solver is constrained to a local minima with only infeasible solutions being presented (e.g., due to excessive glare).

On the current platform, the optimization setup, which is the initialization of JModelica and formation of optimization problem, takes about 45 seconds whereas the actual iterations to find the optimal solution take only 20 to 50 seconds.

6.8. Post-optimization

Once the JModelica optimization is finished, the solver status message is checked for a "successful solution", to verify that the result converged and optimization time was not exceeded. If the solution was not successful, then a new optimization process, with updated input data is started immediately. Otherwise the output of the optimization is passed to the post-optimization stage where the results are converted from a continuous numerical solution to actual control outputs for the dynamic facade.

Usually, and especially in the case of electrochromic windows, dynamic façade components such as motorized shading devices are controlled in discrete control steps whereas the optimization output is a continuous numeric solution which can be any real number. In the simplest case, the output can be rounded to the next integer value. However, this can be problematic, especially with devices with large tint step sizes and/or exponential tint functions, as occurs with EC windows. Rounding to the next integer level, in this case, could lead to increased glare levels.

To account for these cases and to ensure more optimal control, simulations of all available tint positions are conducted before executing the dynamic façade control agent for the next time step. In the case of the three zone electrochromic window, this second set of simulations would involve eight configurations (three zones, three alternate control states), with each electrochromic zone able to be rounded up or down. The results are fed into a simple solver in Python to find the optimal control state for energy minimization while taking glare considerations and limitations on switching of the shading device into account.

Even with these additional simulations, the overall time of conducting the optimization and then running an additional eight simulations is significantly faster (i.e., 15 seconds) and more reliable than adding a penalty function in the main MPC optimization to penalize non-integer values, which would increase the optimization time by 100 seconds.

6.9. Building control

The last step in the optimization is the actual control of the dynamic facade and, if applicable, other building parameters such as room temperature. In the current version, the optimization is designed to communicate with only the dynamic façade device, since the controller is envisioned as being embedded and shipped with the curtainwall unit. Here, communications would rely on the internal communication protocols of the manufactured dynamic façade component; e.g., RS-232, analog signal, Ethernet, or other protocols which are very specific to the actual device. Therefore, in the current configuration, the supervisory building control is embedded in the dynamic facade control agent to allow communication with LBNL's Advanced Windows Testbed via Ethernet/HTTP requests. One future improvement to allow for more flexibility would be the development of a building control agent which could be configured for various communication protocols to enable communication with existing shading devices or the central BMS, as a retrofit solution.

7. Example operation

In the following example, the operation for one control time step with a 24-hour optimization horizon is illustrated. The geometric model describes a single private office (10 x15 x 9 ft) with an electrochromic window oriented due south. The office zone is located in Berkeley, California. The EC window is subdivided into three horizontal control zones, each of which can be controlled to four tint states, from 4-clear (60% Tvis) to 1-dark (1% Tvis), as described earlier.

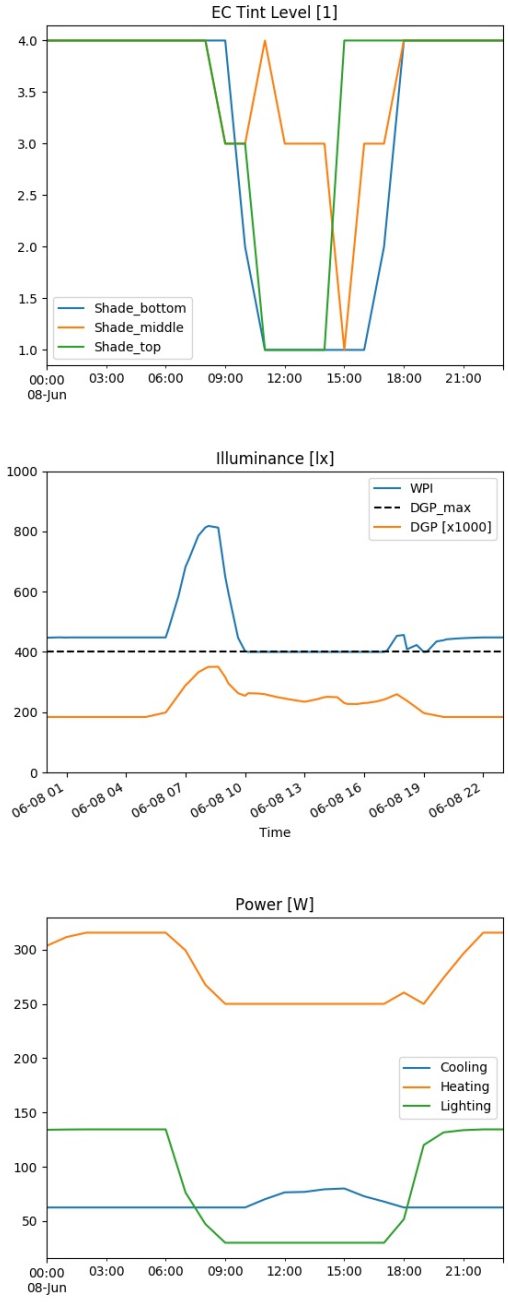


Figure 15. Top graph: EC tint level for the three control zones; middle graph: total workplane illuminance level (lux) and discomfort glare level (DGPs x 1000) in the south-facing zone; lower graph: electricity use for heating, cooling, and lighting on a sunny day, June 8th. The data are a result of control optimization for a single time step and 24-hour optimization horizon.

Figure 15a shows the resulting EC tint levels for each of the three zones for the single control time step. Weather conditions were clear and sunny. The time and date of this control example is June 8 starting at midnight. It can be seen that the bottom control zone, in blue, would be tinted to its darkest state during the middle of the day to reduce solar gains. The top zone, in green, would be remain clear to allow for daylighting, except during the peak hours from 11:00 AM to 2:00 PM, when cooling energy due to solar gains would exceed lighting energy savings from daylighting. The switching pattern of the middle zone, in orange, is the most volatile as it balances solar gains and daylight while maintaining acceptable glare levels.

This balancing is also apparent in the total workplane illuminance (WPI) and glare levels (DGPs), shown in Figure 15b. The workplane illuminance, in blue, is varying as EC tint and outdoor solar radiation levels vary. The setpoint for the electric lighting was a minimum WPI of 400 lx. DGPs, in orange, was set to a maximum (hard constraint) of 0.4 but a soft constraint penalty function (described in Section 6.6) was implemented. Glare was maintained within acceptable levels, even though DGPs reached 0.35 in the morning. Further tuning of the glare penalty function could reduce this spike, but may result in less optimal operation.

Figure 15c shows the actual power necessary to maintain room temperature and the WPI setpoint. It can be seen that the artificial lighting, in green, dims down during the day to the minimal level as daylight increases. Similarly, heating, in orange, and cooling, in blue, is utilized to keep the room temperature at the desired setpoint of 22ºC. It should be noted that minimal levels of heating, 250 W, and cooling, 60 W, were implemented in order to propertly represent the operations of the HVAC systems at the Advanced Windows Testbed.

8. Conclusions

This project has developed an agent-based control system that implements the three-phase matrix approach to determine window heat gains, daylight illuminance, and discomfort glare. This matrix approach enables the simple substitution of BSDF data to model different states of the dynamic façade device, significantly reducing computation time in the derivation of HVAC, lighting, and comfort-related data for integrated façade control. The control system improves upon an approach implemented and tested in a prior CEC PIER project, moving from a solution that relied on a look up table optimized for a fixed grid of conditions to a solution that can now be reconfigured through a web-based user interface (e.g., iPhone, tablet, or computer) and solved in real time. If the occupant changes their direction of view, the control system is updated to perform glare calculations for the new point of view. If the supervisory control system issues a change in electricity cost, the system is able to forecast control over the next 24 hours and respond appropriately. The new control solution provides the end user a powerful new capability to enable autonomous control on a per façade unit basis, tailored to the specific outdoor and indoor conditions of the zone where the façade unit is installed, and coordinated with whole building and grid-level supervisory controls over the life of the installation.

There are two key issues that will ultimately determine feasibility of this solution. 1) The first is the three-way dependency of factors that define the solution: complexity and accuracy of the models, cpu and memory requirements of the hardware platform, and required frequency of the control step for actuation. The simpler the models, the less cpu and memory is required of the hardware, and the faster one can solve for the optimum state for actuation. With less cpu and memory requirements, the cost of the solution is lower. However, simpler models can introduce errors that result in less than optimal control. In this solution, errors are reduced by first using gold standard, complex models to determine performance data related to the strongest variable affecting dynamic façade performance: transmission and distribution of solar radiation. The gold standard three-phase approach produces data for each state of the device. Simple, continuous analytic functions are then fit to these data, which are used in the optimization solver to determine the state for actuation. This coupling of gold standard to simple models enables use of lower-cost hardware and enables more frequent actuation.

2) The second issue is the co-dependency between simplicity of the geometric model that produces the matrices for the three-phase approach and accuracy of the results. The concept of "self-adapting" models will require considerable testing to determine just how much resolution is needed to achieve a desired level of control. For a fast food restaurant with automated shades and a conventional rooftop unit, a generic model may be sufficient to estimate the loads and optimal position of the shades to minimize HVAC energy use. For an office with natural ventilation, a detailed model may be required to ensure that the façade and lighting loads are adequately controlled to ensure comfort throughout a hot summer day. Achieving adaptation and optimization in real-time introduces further challenges.

The initial implementation of the prototype control system demonstrates that the current design is technically feasible, meets the target requirements, but needs more design and engineering to address the above two identified key issues and prove that solution can be robust across a variety of real-world situations.

Acknowledgments

This work was supported by the California Energy Commission through its Electric Program Investment Charge (EPIC) Program on behalf of the citizens of California. LBNL is supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Building Technologies Program of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

Åkesson, J., Arzen, K. E., Gafvert, M., Bergdahl, T., and Tummescheit, H. (2010). Modeling and optimization with Optimica and JModelica.org – Languages and tools for solving large-scale dynamic optimization problems. *Computers and Chemical Engineering*, 34: 1737-1749.

Andersson, J. (2013). A general purpose software framework for dynamic optimization. PhD thesis. Arenberg Doctoral School, KU Leuven.

Blum, D. H., Xu, N., and Norford, L. K. (2016). A novel multi-market optimization problem for commercial HVAC systems providing ancillary services using multi-zone inverse comprehensive room transfer functions. Science and Technology for the Built Environment, 22(6), 783-797.

Blum, D. H., Zakula, T., and Norford, L. K. (2017). Opportunity Cost Quantification for Ancillary Services Provided by Heating, Ventilating, and Air-Conditioning Systems. IEEE Transactions on Smart Grid, 8(3), 1264-1273.

Blum, D. H. and Wetter, M. (2017). MPCPy: An Open-Source Software Platform for Model Predictive Control in Buildings. Proceedings of the 15th Conference of International Building Performance Simulation, Aug 7 – 9, 2017. San Francisco, CA, Accepted.

Bonvini M., Wetter, M., Sohn, M. D. (2014). An FMI-based framework for state and parameter estimation. In proceedings of 10th International Modelica Conference, 2014 - p. 647-656, Lund, Sweden

Braun, J.E. 1990. Reducing energy costs and peak electrical demand through optimal control of building thermal storage. *ASHRAE Transactions* 96(2):876-888.

Coffey, B., Haghighat, F., Morofsky, E., Kutrowski, E. (2010). A software framework for model predictive control with GenOpt. *Energy and Buildings*, 42(7), 1084-1092.

Coffey, Brian E., Andrew McNeil, Thierry Stephane Nouidui, and Eleanor S. Lee. Automated Production of Optimization-Based Control Logics for Dynamic Façade Systems, with Experimental Application to Two-Zone External Venetian Blinds., CEC PIER Technical Report, 2013.

Corbin, C.D., G.P. Henze, and P. May-Ostendorp. 2013. A model predictive control optimization environment for real-time commercial building application. *Journal of Building Performance Simulation* 6(3): 159–174.

De Coninck, R., Magnusson, F., Akesson, J., Helsen, L. (2016). Toolbox for development and validation of grey-box building models for forecasting and control. *Journal of Building Performance Simulation*, 9(3), 288-303.

De Coninck, R., L. Helsen, (2016a). "Practical implementation and evaluation of model predictive control for an office building in Brussels." Energy and Buildings, 111, pp. 290-298.

De Coninck, R., L. Helsen, (2016b). "Quantification of flexibility in buildings by cost curves: Methodology and application." Applied Energy, 162, pp. 653-665.

Dong, B., and Lam, K. P. (2014). A real-time model predictive control for building heating and cooling systems based on the occupancy behavior pattern detection and local weather forecasting. *Building Simulation*, 7(1), 89-106.

Greensfelder, E. M., G. P. Henze, C. Felsmann (2011), "An investigation of optimal control of passive building thermal storage with real time pricing," Journal of Building Performance Simulation, 4:2, pp. 91-104.

Gwerder, M., Gyalistras, D., Sagerschnig, C., Smith R. S., and Sturzenegger, D. (2013). Final Report: Use of Weather And Occupancy Forecasts For Optimal Building Climate Control – Part II: Demonstration (OptiControl-II). Automatic Control Laboratory, ETH Zurich, Switzerland. Available online at: http://www.opticontrol.ethz.ch/Lit/Gwer_13_Rep-OptiCtrl2FinalRep.pdf.

Klems, J.H., 1994a. A new method for predicting the solar heat gain of complex fenestration systems: I. Overview and derivation of the matrix layer calculation, *ASHRAE Transactions* 100 (1): 1065-1072.

Li, P., D. Vrabie, D. Li, C.B. Sorin, S. Mijanovic, and Z.D. O'Neill. 2015. Simulation and experimental demonstration of model predictive control in a building HVAC system. *Science and Technology for the Built Environment* 21(6): 721-732.

Mattson, S. E. and Elmqvist, H. (1997). Modelica – An international effort to design the next generation modeling language. In 7th IFAC Symposium On Computer Aided Control Systems Design, Gent, Belgium, April 28-30.

Magnusson, F., and Åkesson, J. (2012). Collocation Methods for Optimization in a Modelica Environment. 9th International Modelica Conference, Munich, Germany.

Mitchell, R. Kohler, C., Klems, J., Rubin, M., Arasteh, D., Huizenga, C., Yu, T. Curcija, D., 2008. Window 6.2/ Therm 6.2 Research Version User Manual, Lawrence Berkeley National Laboratory, LBNL-941. Available http://windows.lbl.gov/software/window/window.html (accessed April 15, 2016).

NREL, 2017, https://nsrdb.nrel.gov/tmy (accessed July 17, 2017)

Oldewurtel, F., A. Ulbig, A. Parisio, G. Andersson, and M. Morari, "Reducing Peak Electricity Demand in Building Climate Control using Real-Time Pricing and Model Predictive Control," 49th IEEE Conference on Decision and Control, Dec. 2010, pp. 1927-1932.

Pavlak, G.S., G.P. Henze, and V.J. Cushing. 2014. Optimizing commercial building participation in energy and ancillary service markets. *Energy and Buildings* 81: 115-126.

Qin, S., and T. Badgwell (2003). A survey of industrial model predictive control technology. *Control Engineering Practice*, 11(7), 733-764.

Raspberry Pi, https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

Sturzenegger, D., Gyalistras, C., and Morari, M. (2016). Model predictive control of a swiss office building: Implementation, results, and cost-benefit analysis. *IEEE Transactions on Control Systems Technology*, 24(1),

Vrettos, E., F. Oldewurtel, F. Zhu, and G. Andersson (2014). "Robust provision of frequency reserves by office building aggregations," World Congress of the International Federation of Automatic Control (IFAC), 2014, pp. 12068-12073.

Ward, G., Mistrick, R., Lee, E.S., McNeil, A. Jonsson, J., 2011. Simulating the Daylight Performance of Complex Fenestration Systems Using Bidirectional Scattering Distribution Functions within Radiance. *Leukos* 7(4): 241-261.

Ward Larson, G. and Shakespeare, R., 1998, Rendering with Radiance: The Art and Science of Lighting Visualization. San Francisco: Morgan Kaufmann.

Wetter, M., Bonvini, M., and Nouidui, T. S. (2016). Equation-based languages – A new paradigm for building energy modeling, simulation and optimization. *Energy and Buildings*, 117, 290-300.

Wienold, J., Christoffersen, J., 2006. Evaluation methods and development of a new glare prediction model for daylight environments with the use of CCD cameras. *Energy and Buildings* 38 (7): 734-757.

Wienold, J., 2009. Dynamic daylight glare evaluation, Building Simulation 2009, 11th International IBPSA Conference, Glasgow, Scotland, July 27-30.

Zakula, T., P.R. Armstrong, and L. Norford. 2015. Advanced cooling technology with thermally activated building surfaces and model predictive control. *Energy and Buildings* 86: 640-650.

Appendix A. Web-based interface